NUM3 sur owncloud

https://cloud-soc.lip6.fr/owncloud/public.php?service=files&t=19a15e3a0fff58e8b260c16fc6e54605

Electronique numérique 3 : Introduction

Patrick.Garda@upmc.fr

UPMC SORBONNE UNIVERSITÉS

POLYTECH PARIS-UPMC

Objectifs & Organisation

- Conceptuels
  - Conception circuits électroniques numériques en VHDL
  - Machines à Etat
  - Microarchitecture
  - Composants : FPGA
- Opératoires
  - Logiciel simulation et synthèse
  - Prototypage sur carte FPGA ALTERA
- Faire le lien
  - NUM1, NUM2, ARCHI, CODESIGN

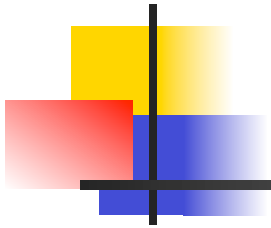
- Enseignement
  - 5 x 2h cours
  - 5 x 4h TP
- Contrôle des connaissances
  - Moyenne 4x1/2h CC
  - Moyenne TP
- Ressources pour l'UE
  - Owncloud

Electronique numérique 3 : micro-architecture d'un processeur mono-cycle

Patrick.Garda@upmc.fr

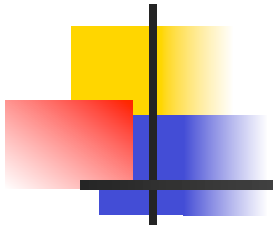
UPMC SORBONNE UNIVERSITÉS

POLYTECH PARIS-UPMC



Introduction

- Objectif
  - Proposer une méthode de conception de fonctions électroniques numériques
- Moyen
  - Exemple d'un cœur de processeur
- Source
  - Prof. James L. Johnson
  - Computer Science Dept
  - Western Washington University
  - http://faculty.cs.wwu.edu/johnson/



How to Design a Processor: step-by-step

1. Analyze instruction set => datapath requirements
   a. the meaning of each instruction is given by the *register transfers*
   b. datapath must include storage element for ISA registers
      and possibly some intermediate results
   c. datapath must support each register transfer

2. Select set of datapath components and establish clocking methodology

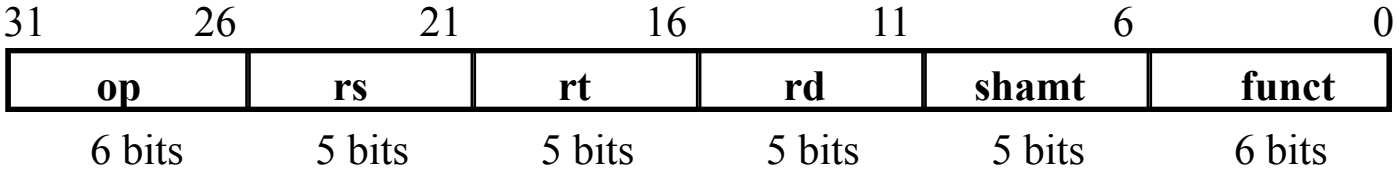
3. Assemble datapath meeting the requirements

4. Analyze implementation of each instruction to determine setting of
   control points that effects the register transfer.

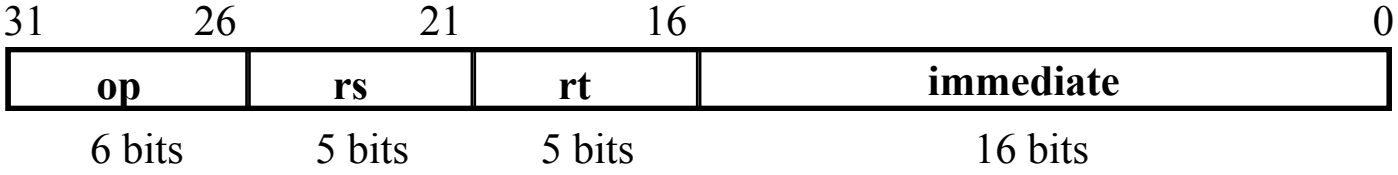
5. Assemble the control logic

The MIPS Instruction Formats

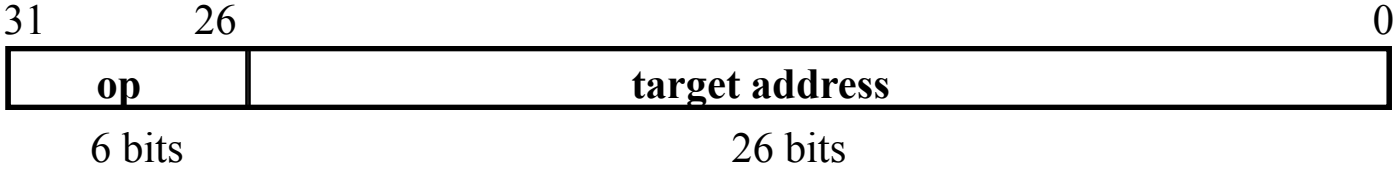
- All MIPS instructions are 32 bits long. The three instruction formats:
  - R-type



  - I-type



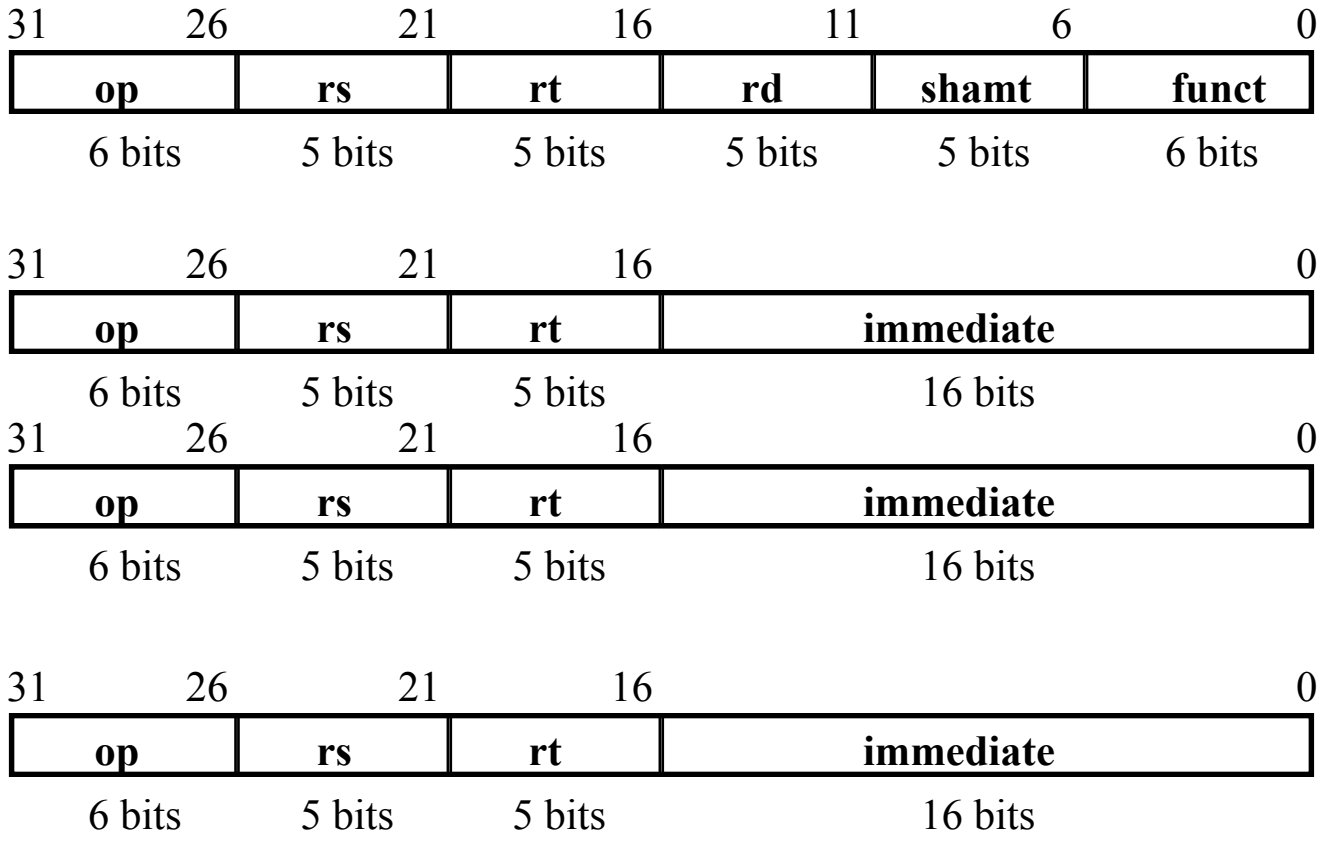
  - J-type



- The different fields are:
  - op: operation
  - rs, rt, rd: source and destination register
  - shamt: shift amount
  - funct: selects operation variant
  - address / immediate: address offset or immediate value
  - target address: target address of the jump instruction

Step 1a: The MIPS-lite Subset

- add, sub, and, or, slt; e.g.
  - add rd, rs, rt
  - sub rd, rs, rt
- immediate; e.g.
  - ori rt, rs, imm16
- memory reference; e.g.
  - lw rt, rs, imm16
  - sw rt, rs, imm16
- branch:
  - beq rs, rt, imm16



Instruction definition

- RTL gives the meaning of the instructions
- All start by fetching the instruction

op | rs | rt | rd | shamt | funct = MEM[PC]

op | rs | rt | Imm16 = MEM[PC]

inst	Register Transfers	
ADD	R[rd] <– R[rs] + R[rt];	PC <– PC + 4
SUB	R[rd] <– R[rs] – R[rt];	PC <– PC + 4
ORI	R[rt] <– R[rs] | zero_ext(Imm16);	PC <– PC + 4
LOAD	R[rt] <– MEM[R[rs] + sign_ext(Imm16)];	PC <– PC + 4
STORE	MEM[R[rs] + sign_ext(Imm16)] <– R[rt];	PC <– PC + 4

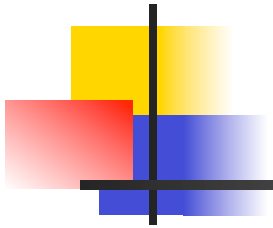
BEQ		if (R[rs] == R[rt])
		PC <- PC + 4 + (SignExt(imm16) x 4)
		else
		PC <– PC + 4

I – Chemin de données datapath

Patrick.Garda@upmc.fr

UPMC SORBONNE UNIVERSITÉS

POLYTECH PARIS-UPMC

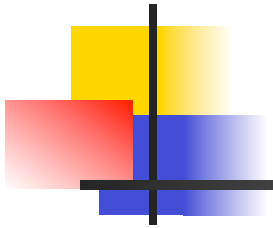


Step 1b: ISA storage elements

- Memory
  - instruction & data
- Registers (32 x 32)
  - read RS
  - read RT
  - Write RT or RD
- PC
- SR

Step 1c: ISA operations

- Extender
- Add and Sub register or extended immediate
- Add 4 or extended immediate to PC
- OR Bitwise

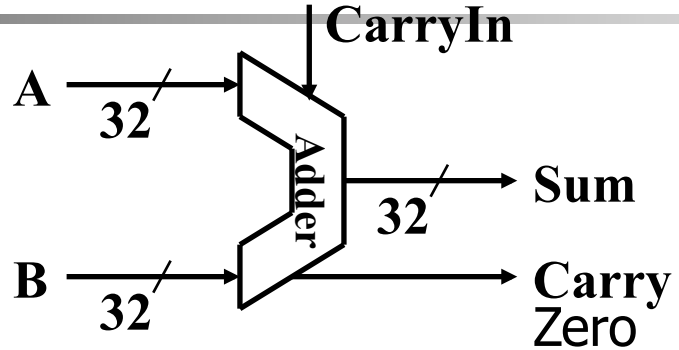


Step 2: Components of the Datapath

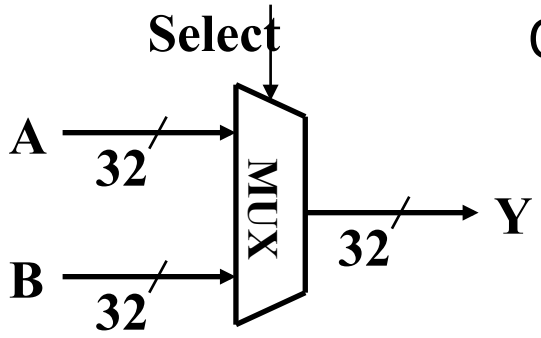
- Combinational Elements
- Storage Elements
  - Clocking methodology

Combinational Logic Elements

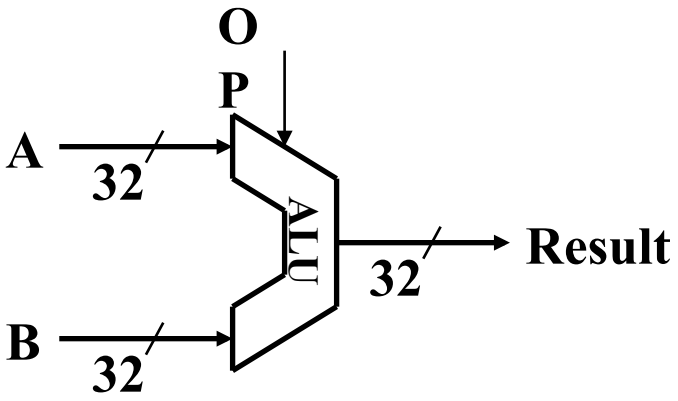
- Adder



- MUX

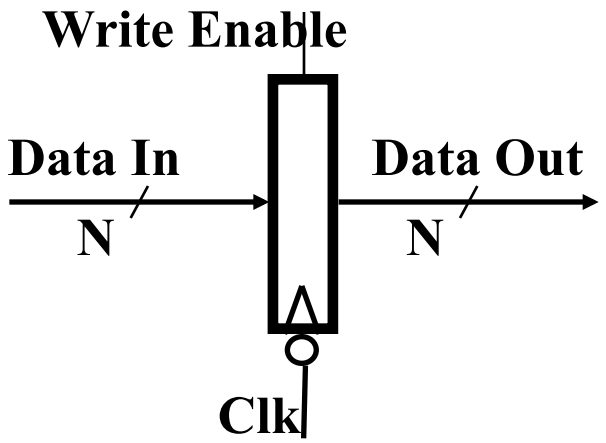


- ALU



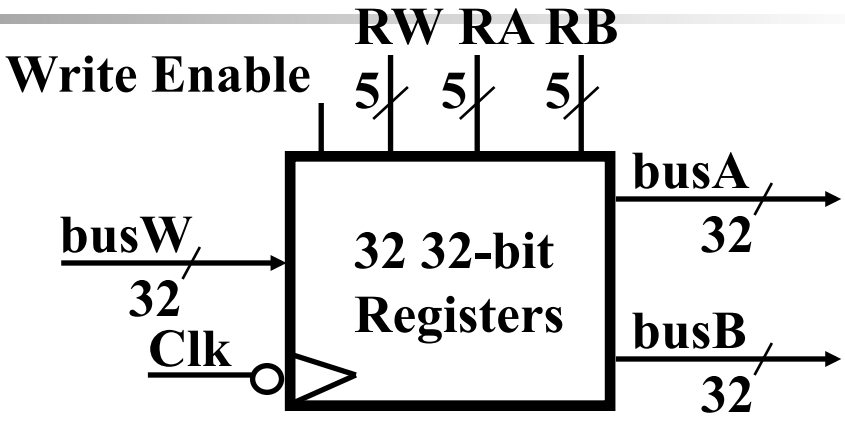
Storage Element: Register

- Register
  - Similar to the D Flip Flop except
    - N-bit input and output
    - Write Enable input
  - Write Enable:
    - negated (0): Data Out will not change
    - asserted (1): Data Out will become Data In



Storage Element: Register File

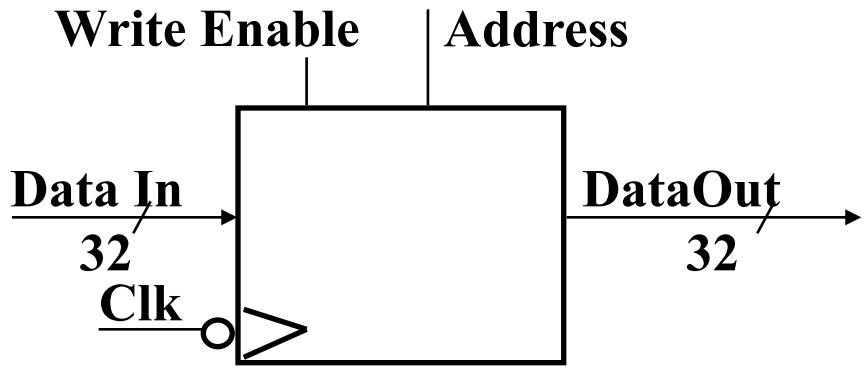
- Register File consists of 32 registers:
  - Two 32-bit output busses:
    busA and busB
  - One 32-bit input bus: busW



- Register is selected by:
  - RA (number) selects the register to put on busA (data)
  - RB (number) selects the register to put on busB (data)
  - RW (number) selects the register to be written from busW (data)
    when Write Enable is 1

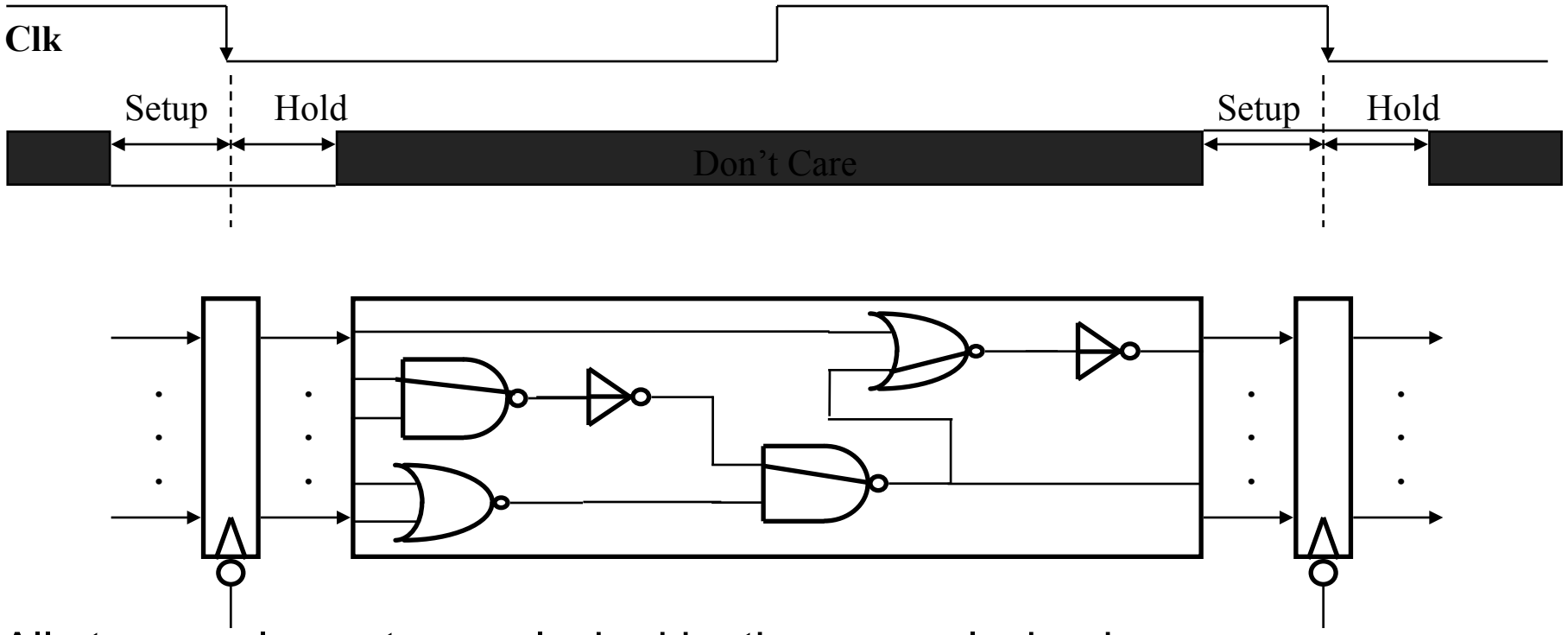
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - RA or RB valid => busA or busB valid after “access time.”

Storage Element: Idealized Memory



- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is selected by:
  - Address selects the word to put on Data Out
  - Write Enable = 1: address selects the memory word to be written via the Data In bus
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
  - Address valid => Data Out valid after “access time.”

Clocking Methodology



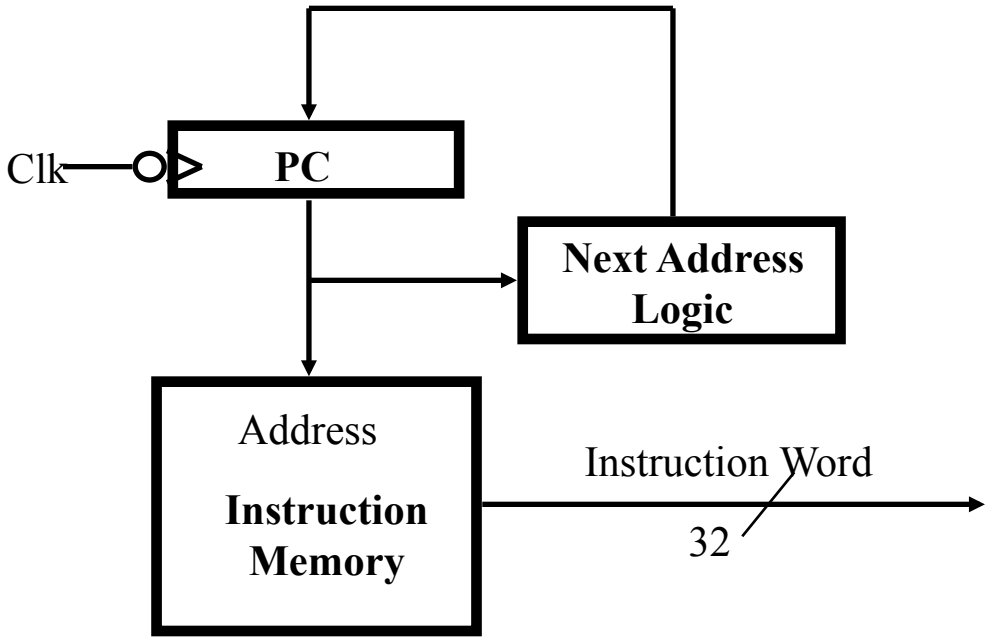
- All storage elements are clocked by the same clock edge
- Cycle Time = CLK-to-Q + Longest Delay Path + Setup + Clock Skew
- (CLK-to-Q + Shortest Delay Path - Clock Skew) > Hold Time

Step 3

- Register Transfer Requirements –> Datapath Assembly
- Instruction Fetch
- Read Operands and Execute Operation

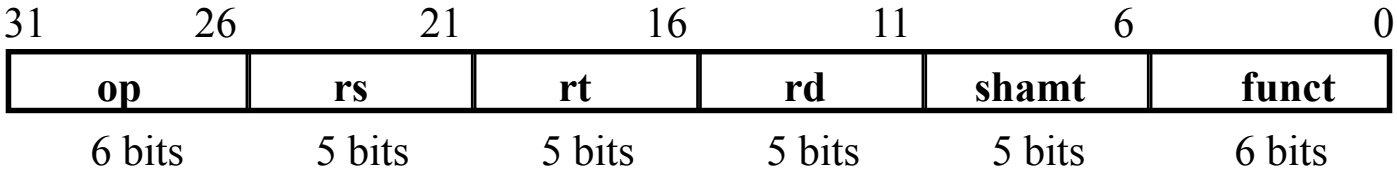
3a: Overview of the Instruction Fetch Unit

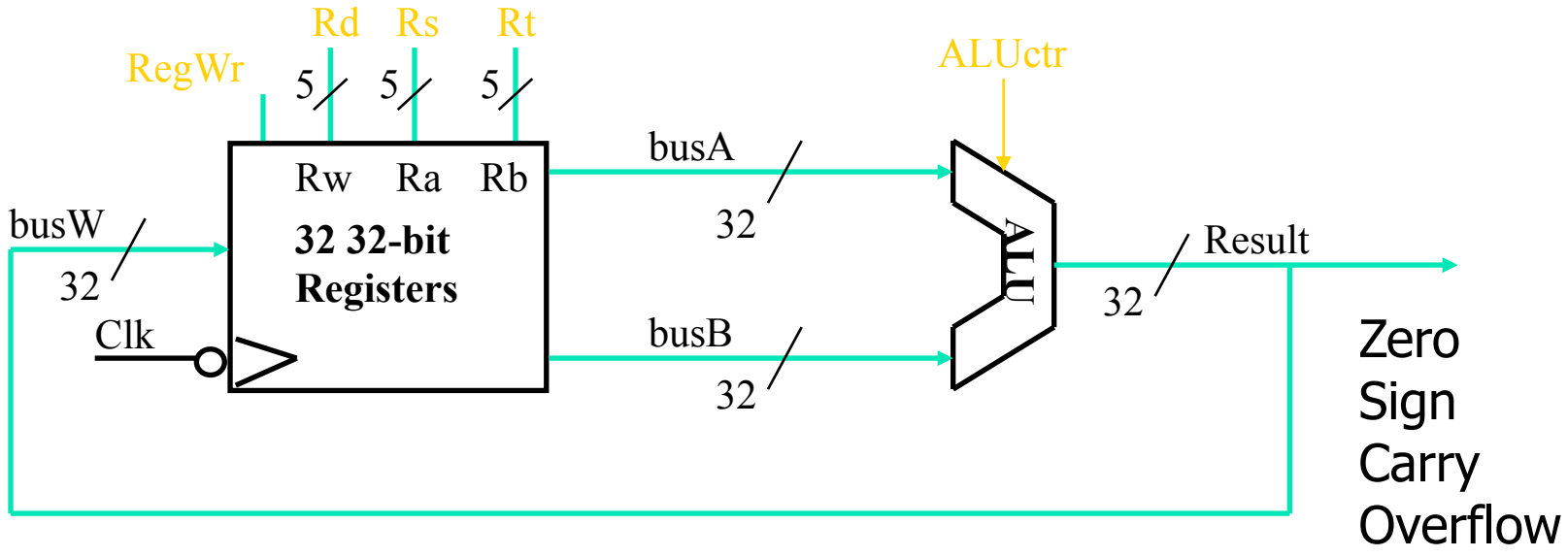
- The common RTL operations
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC <- PC + 4
    - Branch and Jump: PC <- “something else”



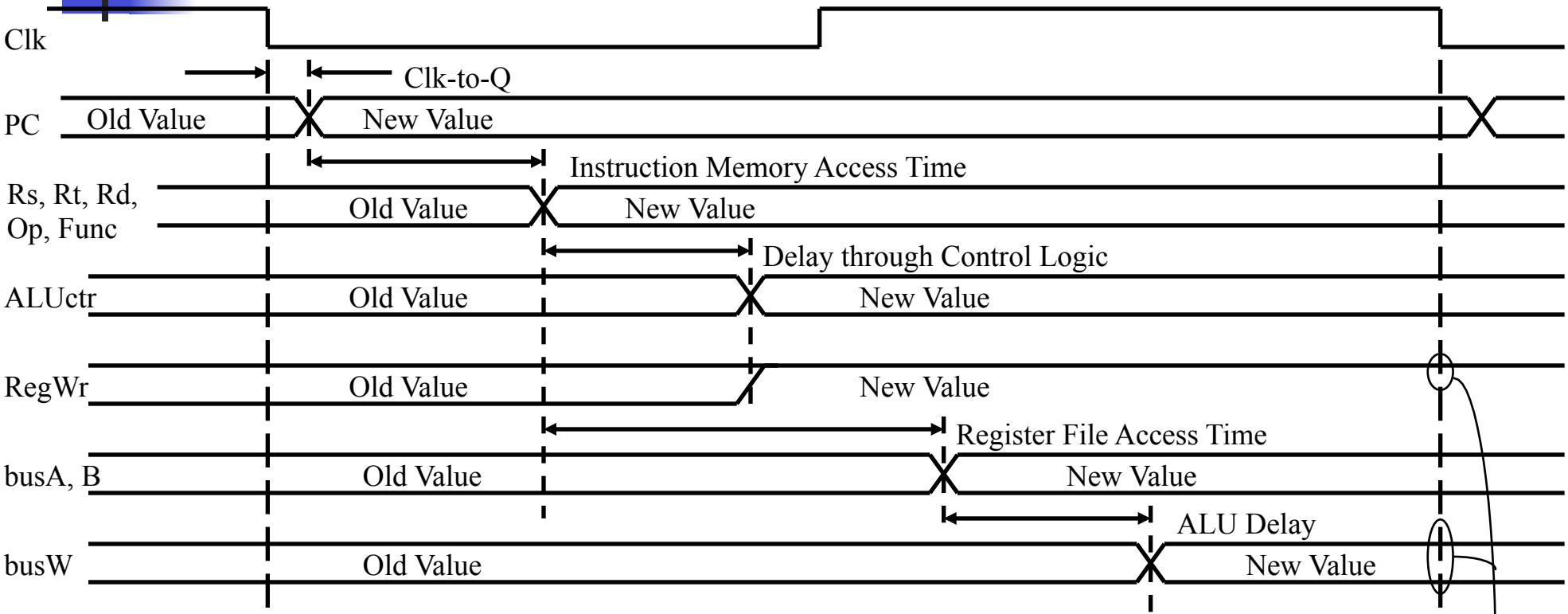
3b: Add & Subtract

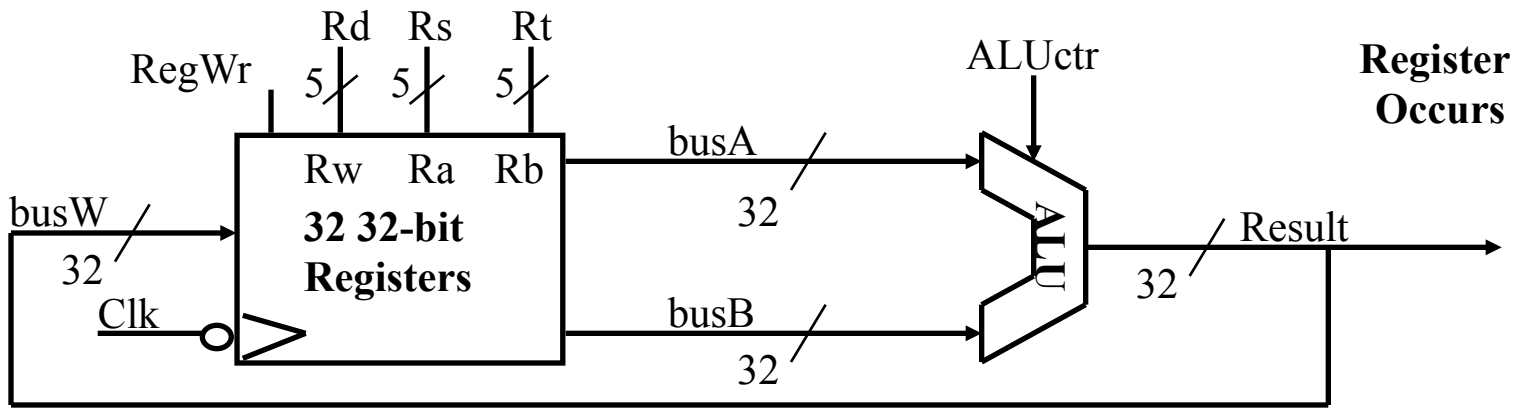
- R[rd] <- R[rs] op R[rt] Example: add rd, rs, rt
  - Ra, Rb, and Rw come from instruction's rs, rt, and rd fields
  - ALUctr and RegWr: control logic after decoding the instruction





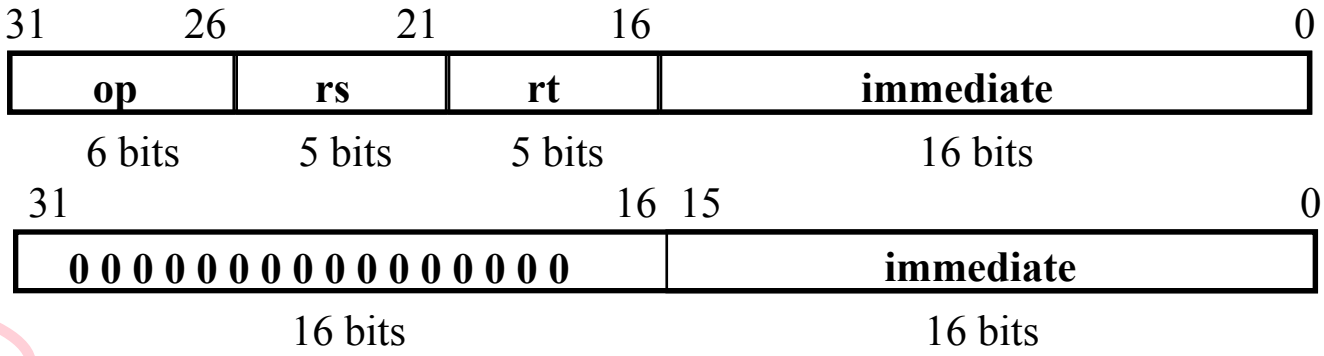
Register-Register Timing

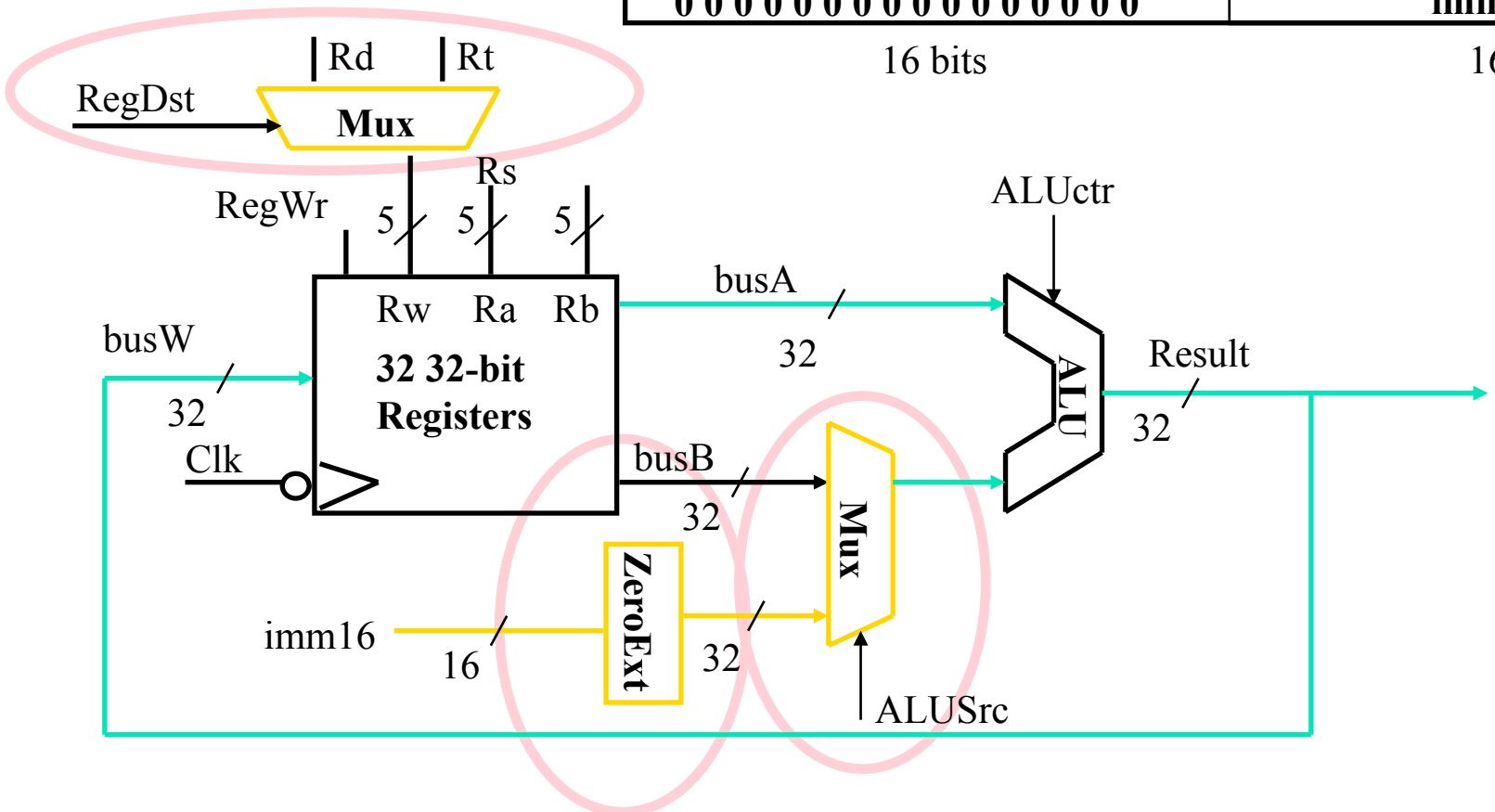


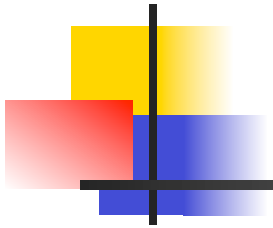


3c: Logical Operations with Immediate

- R[rt] <- R[rs] ORI ZeroExt[imm16]]





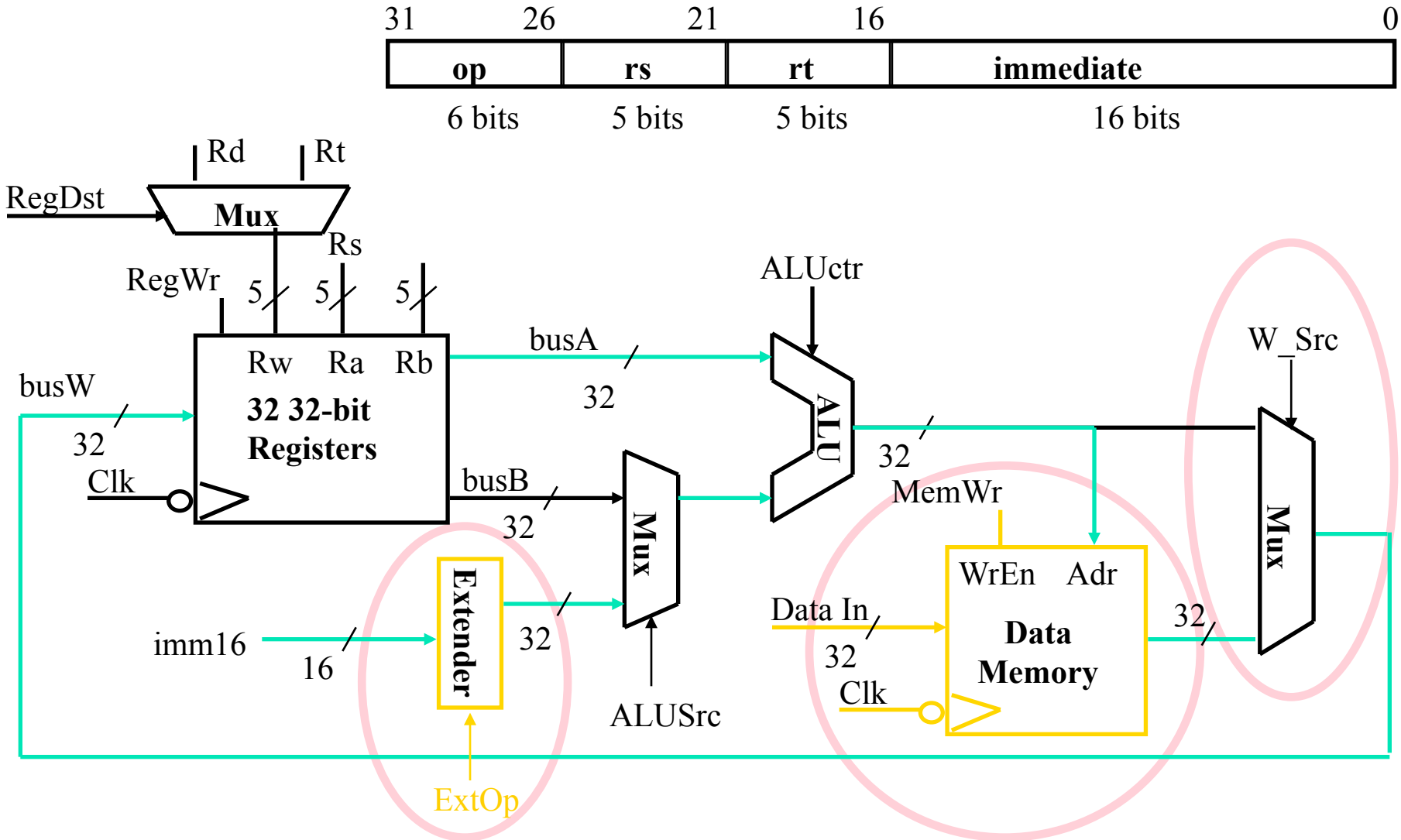


C2

- - b15 2^15 + b14 2^14 + …
- = (- 2 + 1) b15 2^15 + b14 2^14 + …
- = - b15 2^16 + b15 2^15 + b14 2^14 + …

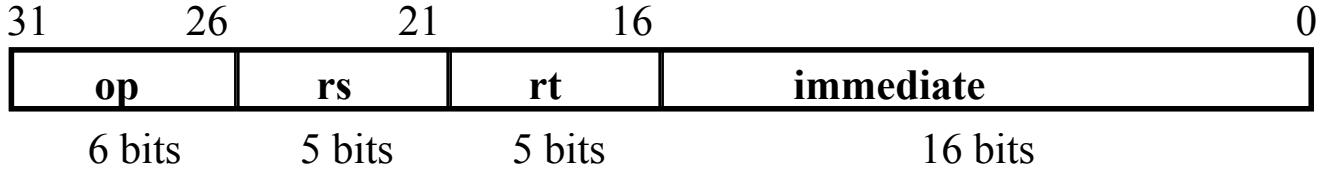
3d: Load Operations

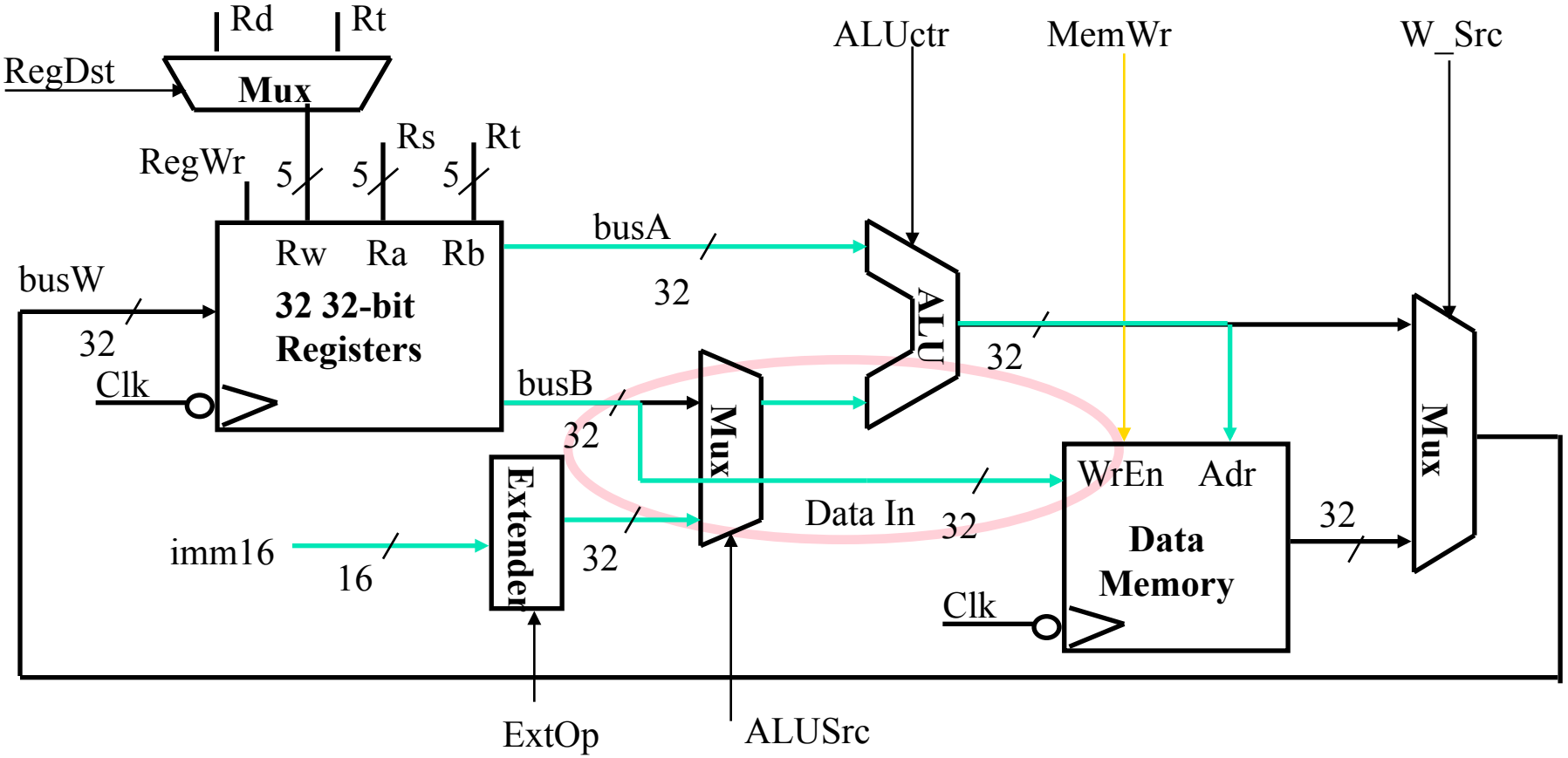
- R[rt] <- Mem[R[rs] + SignExt[imm16]]



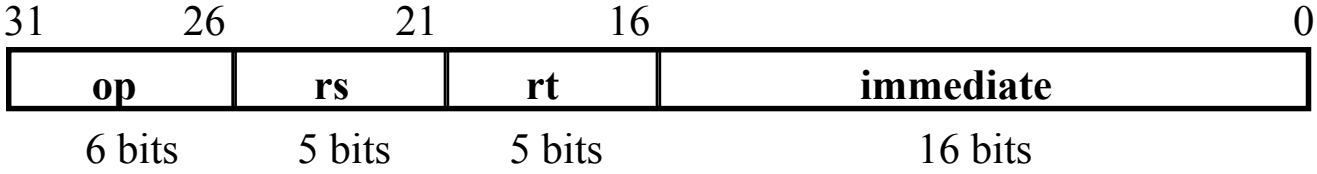
3e: Store Operations

- Mem[R[rs] + SignExt[imm16]] <- R[rt]





3f: The Branch Instruction

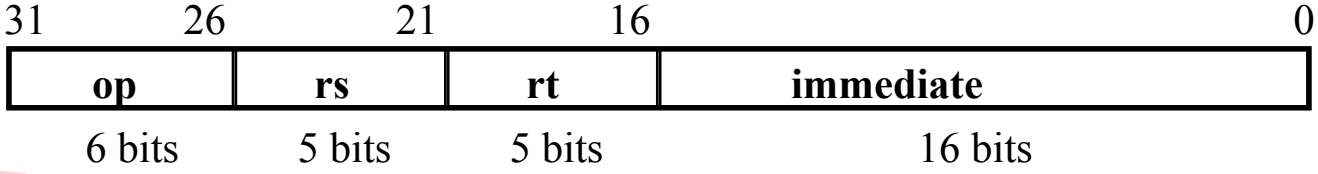


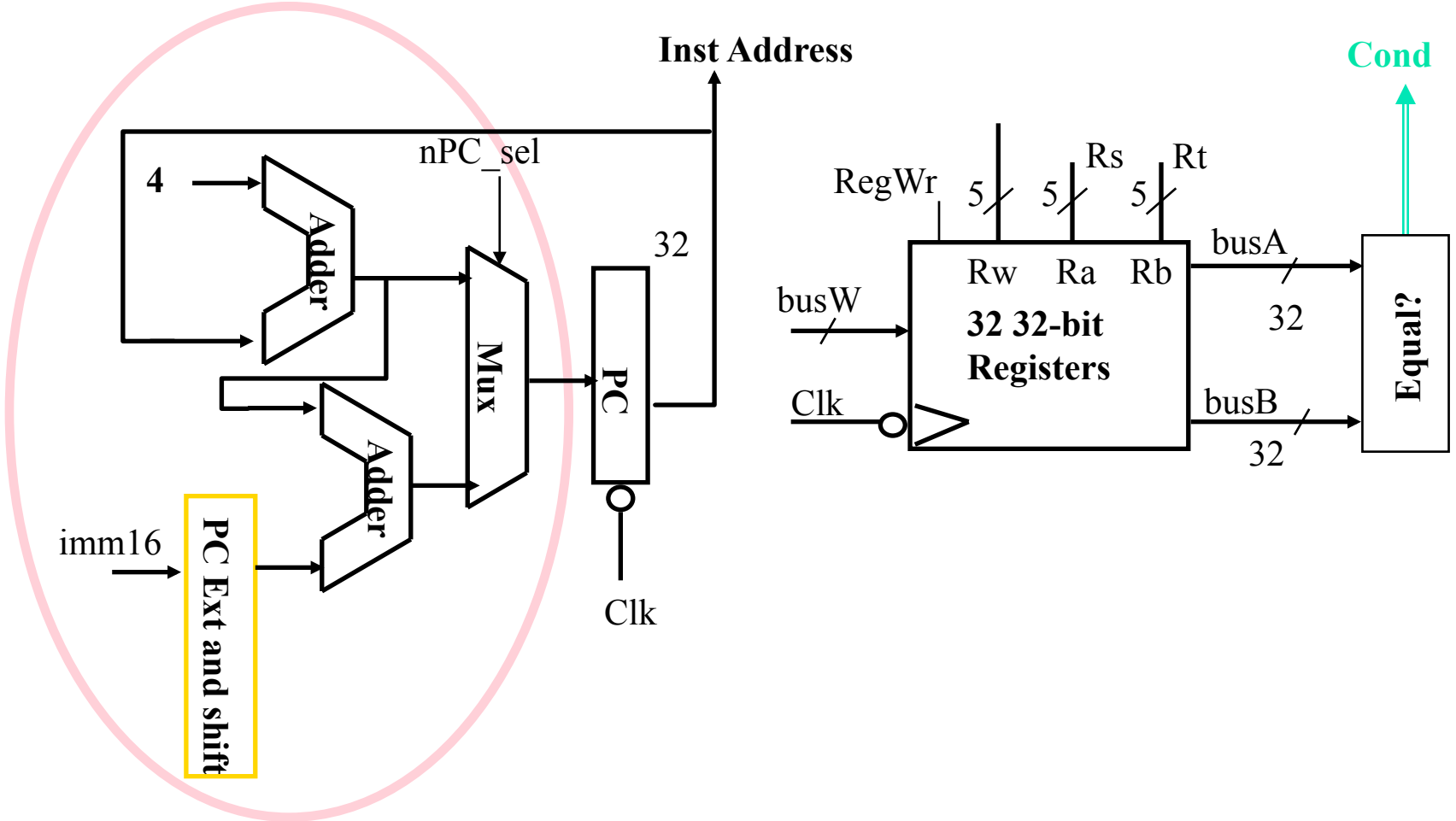
- beq rs, rt, imm16
  - mem[PC] Fetch the instruction from memory
  - Equal <- R[rs] == R[rt] Calculate the branch condition
  - if (COND eq 0) Calculate the next instruction's address
    PC <- PC + 4 + (SignExt(imm16) x 4)
  - else
    PC <- PC + 4

Datapath for Branch Operations

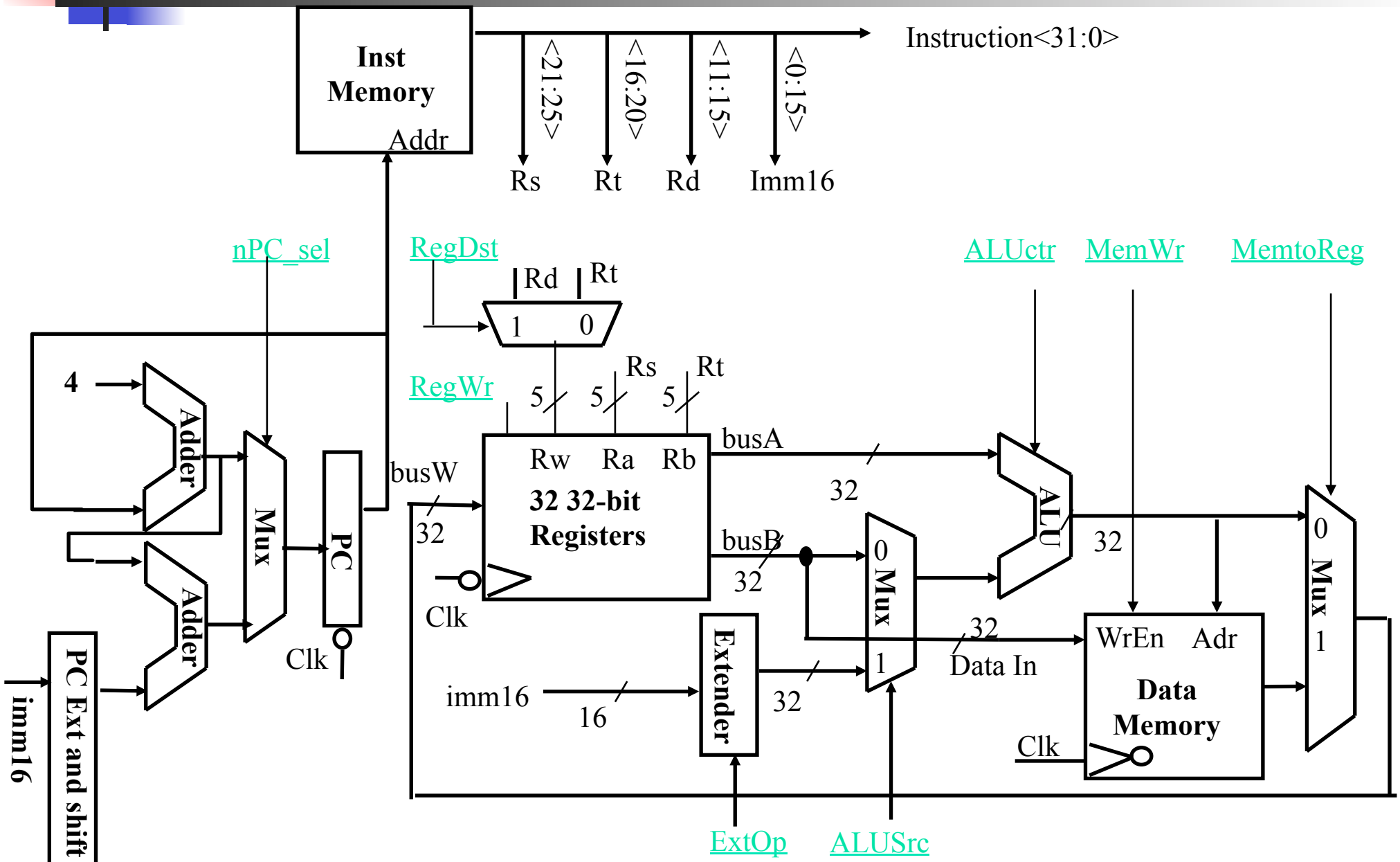
- beq rs, rt, imm16

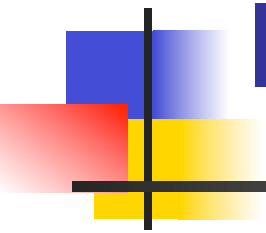
Datapath generates condition (equal)





Putting it All Together: A Single Cycle Datapath





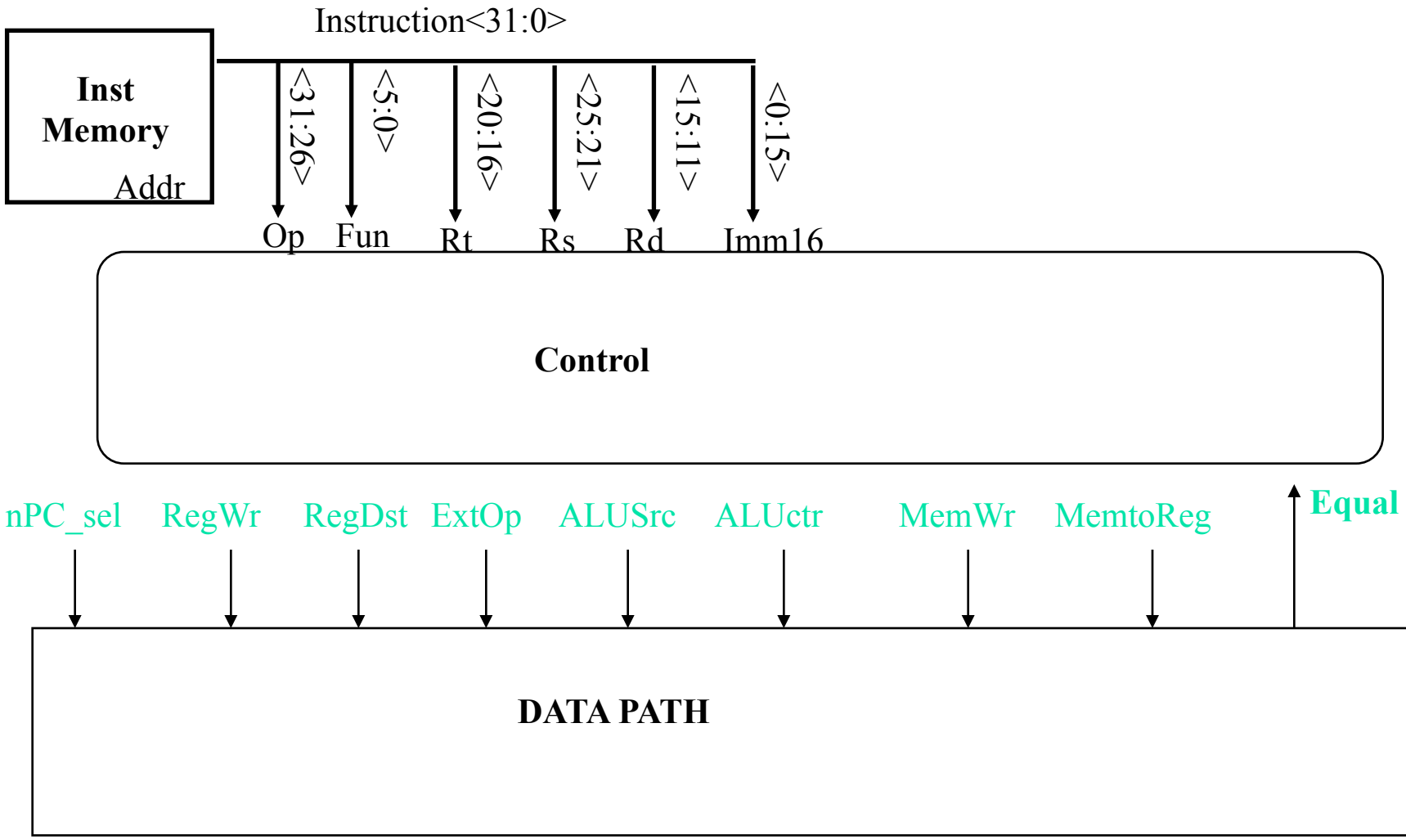
II - Contrôle

Patrick.Garda@upmc.fr

UPMC SORBONNE UNIVERSITÉS

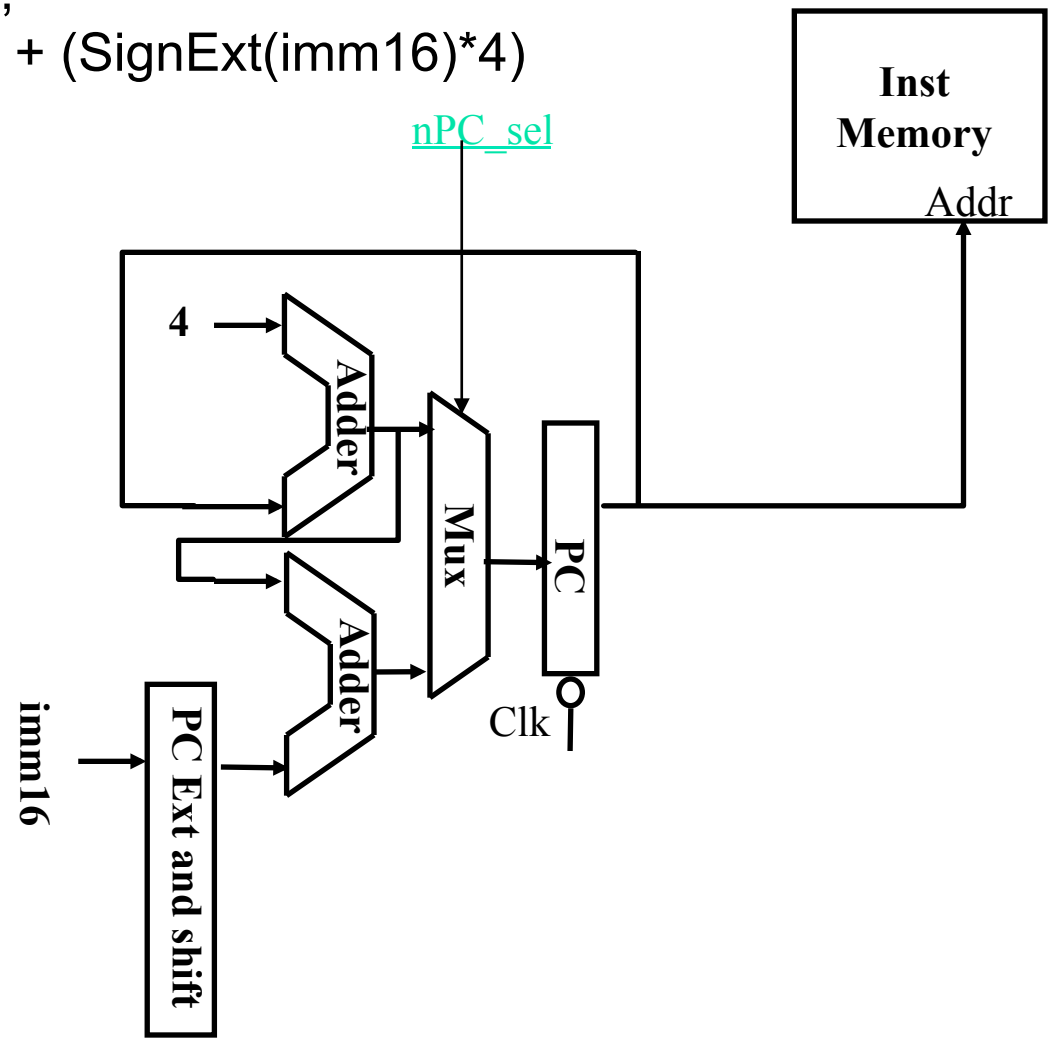
POLYTECH PARIS-UPMC

Step 4: Datapath, RTL -> Control



Meaning of the Control Signals

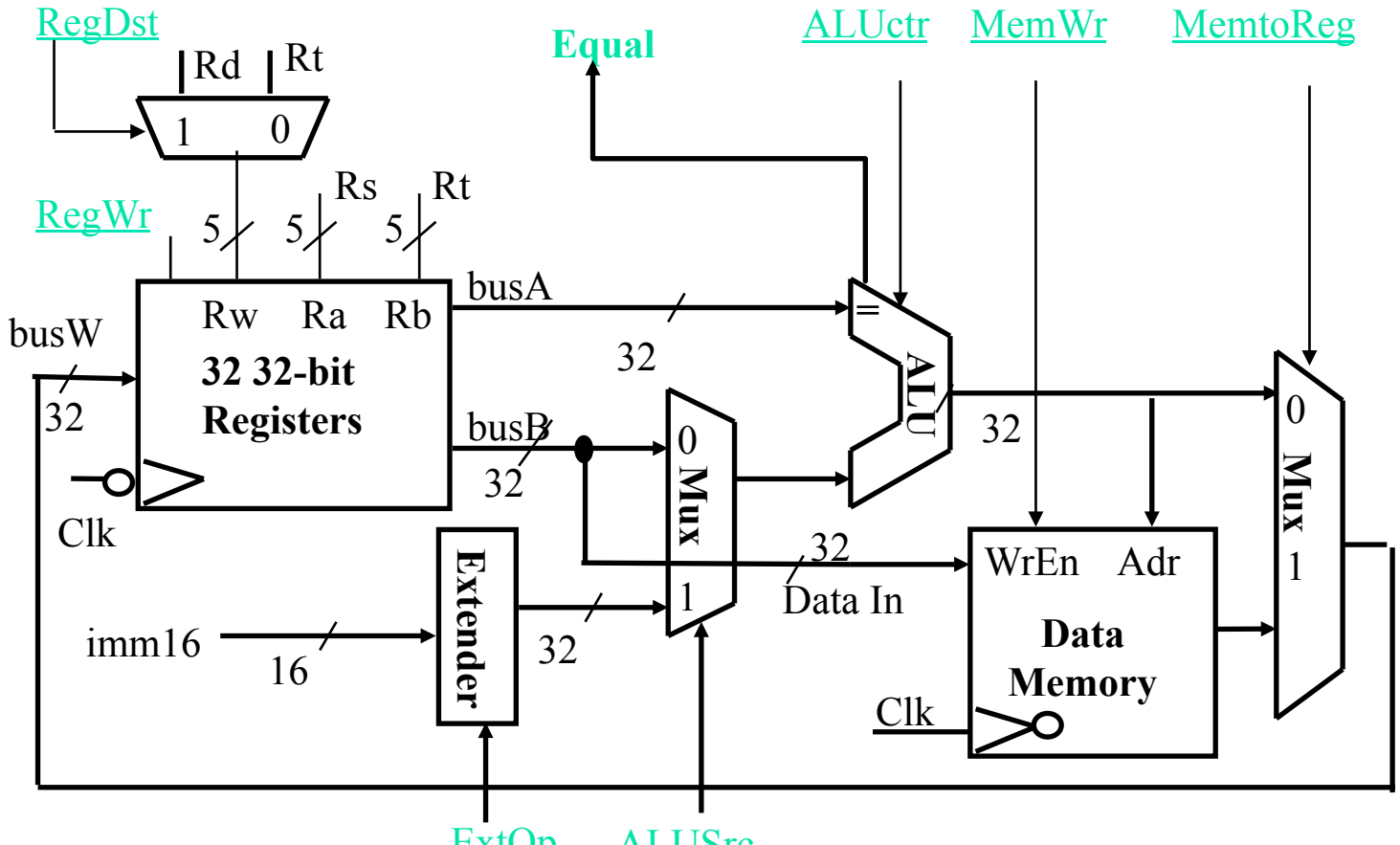
- Rs, Rt, Rd and imm16 hardwired into datapath
- nPC_sel: 0 => PC <– PC + 4;
  1 => PC <– PC + 4 + (SignExt(imm16)*4)



Meaning of the Control Signals

- ExtOp: "zero", "sign"
- ALUsrc: 0 => regB; 1 => immed
- ALUctr: "add", "sub", "or"

- **MemWr: write memory**
- **MemtoReg: 1 => Mem, 0 => ALU out**
- **RegDst: 0 => "rt"; 1 => "rd"**
- **RegWr: write dest register**



Control Signals to accomplish RTL

inst	Register Transfer	
ADD	R[rd] <– R[rs] + R[rt];	PC <– PC + 4
	ALUsrc = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”	
SUB	R[rd] <– R[rs] – R[rt];	PC <– PC + 4
	ALUsrc = RegB, ALUctr = “sub”, RegDst = rd, RegWr, nPC_sel = “+4”	
ORI	R[rt] <– R[rs] + zero_ext(Imm16);	PC <– PC + 4
	ALUsrc = Imm, Extop = “Zero”, ALUctr = “or”, RegDst = rt, RegWr, nPC_sel = “+4”	
LOAD	R[rt] <– MEM[R[rs] + sign_ext(Imm16)];	PC <– PC + 4
	ALUsrc = Imm, Extop = “Sign”, ALUctr = “add”, MemtoReg, RegDst = rt, RegWr, nPC_sel = “+4”	
STORE	MEM[R[rs] + sign_ext(Imm16)] <– R[rs];	PC <– PC + 4
	ALUsrc = Imm, Extop = “Sign”, ALUctr = “add”, MemWr, nPC_sel = “+4”	

BEQ	if (R[rs] == R[rt])
	PC <- PC + 4 + (SignExt(imm16) x 4)
	else
	PC <– PC + 4	

	nPC_sel = EQUAL, ALUctr = “sub”	

Step 5: Logic for each control signal

- nPC_sel <= if (OP = BEQ) then EQUAL else 0 endif
- ALUsrc <= if (OP = “000000”) then “regB” else “immed” endif
- ALUctr <= if (OP = “000000”) then **funct**
  elsif (OP = ORI) then “OR”
  elsif (OP = BEQ) then “sub”
  else “add”
  endif
- ExtOp <= if (OP = ORI) then “zero” else “sign” endif
- MemWr <= (OP = Store)
- MemtoReg <= (OP = Load)
- RegWr: <= if ((OP = Store) OR (OP = BEQ)) then 0 else 1 endif
- RegDst: <= if ((OP = Load) OR (OP = ORI)) then 0 else 1 endif

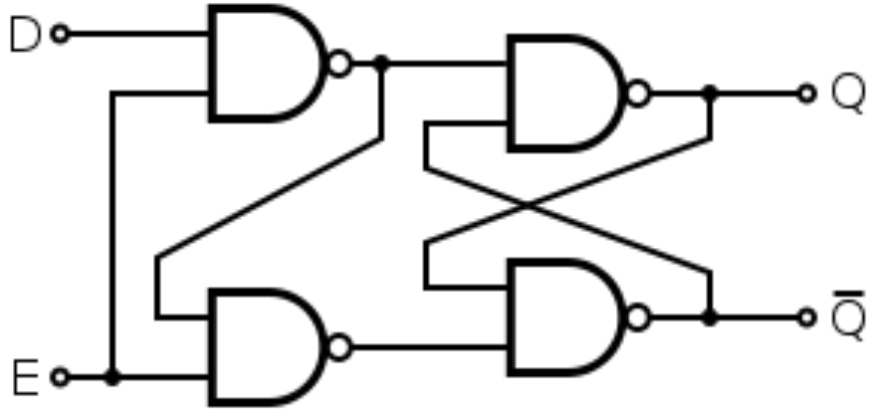
Bascules : temps de prépositionnement et de maintien

Patrick.Garda@upmc.fr

UPMC SORBONNE UNIVERSITÉS

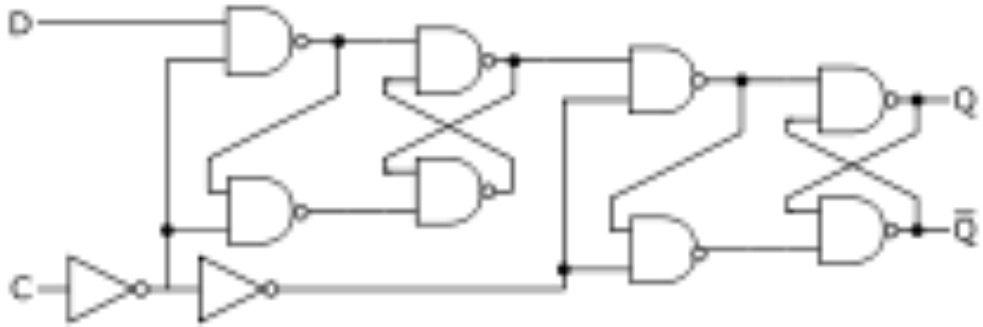
POLYTECH PARIS-UPMC

Latch D



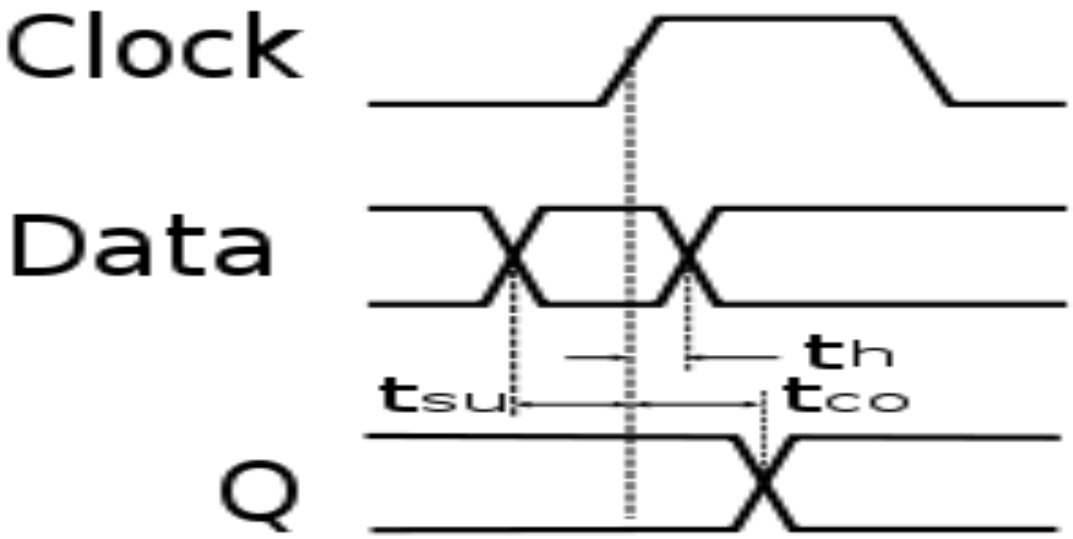
- Si E = 0 Latch verrouillé
- Si E = 1 Latch transparent

Bascule D



- Si C = 0
  - Latch Maître transparent
  - Latch Esclave verrouillé
- Si C = 1
  - Latch Maître verrouillé
  - Latch Esclave transparent

Temps prépositionnement



- Tsu
  - Temps prépositionnement
- Th
  - Temps maintien
- Tco Tc-to-Q Tp
  - Temps propagation

Période du processeur monocycle

Patrick.Garda@upmc.fr

UPMC SORBONNE UNIVERSITÉS

POLYTECH PARIS-UPMC

Worst Case Timing (Load instruction)